Completeness Estimation of Range Queries in Structured Overlays*

Marcel Karnstedt, Kai-Uwe Sattler
Faculty of Computer Science and Automation
TU Ilmenau, Germany
{marcel.karnstedt|kus}@tu-ilmenau.de

Roman Schmidt
Ecole Polytechnique Fédérale
de Lausanne (EPFL), Switzerland
Roman.Schmidt@epfl.ch

Abstract

Range queries are a very powerful tool in a wide range of data management systems and are vital to a multitude of applications. The hierarchy of structured overlay systems can be utilized in order to provide efficient techniques for processing them, resulting in the support of applications and techniques based on range queries in large-scale distributed information systems. But, due to the limited knowledge and the usually best-effort characteristics, deciding about the completeness of query results, e.g., getting an idea when a query is finished or what amount of results is still missing, is very challenging. There is not only an urgent need to provide this information to the user issuing queries, but also for implementing sophisticated and efficient processing techniques based on them. In this work, we propose a method for solving this task. We discuss the applicability and quality of our estimations, present an implementation and evaluation for the P-Grid system and show how to adapt the technique to other overlays.

1. Introduction

P2P systems and particularly structured overlays based on distributed hashtables (DHTs) are recognized as a promising infrastructure for large-scale distributed data management. The main reasons are their effectiveness and scalability as well as the predictable behavior. After the first generation supporting only basic key lookups, recent research efforts address also the problem of efficiently querying range predicates [5, 8]. Typically, these approaches exploit the structure of the overlay (e.g., a tree structure) by implementing some kind of multicast protocol. In this context, a main challenge is to estimate the progress of query processing, i.e., to answer the question which fraction of the total query result is already received. The difficulties are due to the purely decentralized nature of the structured overlay, the lack of global knowledge (no peer knows how many peers are responsible for the queried key range), the dynamics of the network (peers may leave the network during processing a query), as well as the often used best-effort strategy for query routing and answering.

However, estimating the completeness of a query result is not only a helpful information for the user issuing the query, but it is also needed for processing complex queries. For instance, query operators like aggregation or ranking-based queries (e.g., skyline queries [6, 10]) require to know when all input data is arrived in order to calculate the aggregate value or to sort the input.

In this paper, we propose a solution to this problem. The objective of our work is to estimate the completeness of range queries as a fundamental operator for more complex query operators and to give guarantees on the quality of this estimation. The idea is to map the completeness on data level to a completeness on peer level, thus, estimating a number of replies expected for each query. This, of course, is still very challenging in the investigated systems. We have implemented the proposed approach in our UniStore system [11] which is based on the P-Grid overlay. But we show also how the main idea can be applied to other DHTs, too. UniStore supports complex structured database-like queries and thus, strongly benefits from a method for estimating query completeness. For instance, aggregation queries and skyline queries internally rely on range queries. Results of such queries can be processed after a satisfyingly significant portion of the result data (e.g., 90%) is received, rather than demanding for complete result sets. Due to the usually parallelized processing of range queries in DHT overlays, the decision whether a result set is complete or satisfyingly large can only be made using a sophisticated method for completeness estimation as proposed in this work. This will not save bandwidth or processing power, as later replies are still transmitted, but eliminates the need for static waiting states and enables speedy processing of subsequent operators of complex queries.

The remainder of this paper is structured as follows: Section 2 briefly discusses approaches for processing range queries in overlay systems and introduces the technique pro-

*The work presented in this paper was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project NEPOMUK No FP6-027705.

posed in P-Grid. The actual approach for completeness estimation based on this is presented in Section 3. Afterwards, we discuss the technique's applicability for other systems in Section 4 and explain the user point of view on completeness estimation in Section 5. Finally, we conclude the paper by an evaluation in Section 6 and a summary.

2. Range Queries in Structured Overlays

Range queries were proposed in the past for several structured overlays [8], [5], [4, 3], [14, 13], [9, 15]. Like UniStore, [7] proposes methods for range queries and advanced query types on DHT overlays. But, this work lacks discussion about estimating query completeness. We base our work on the approach presented in [8] using P-Grid [1] as structured overlay network. P-Grid organizes nodes in a binary trie structure, which is a standard indexing structure from databases to support among others range queries. A key benefit of tries is that they cluster semantically close data items which is a critical pre-condition for efficient processing of range queries. Traditional DHTs such as Chord or Pastry use uniform hashing functions to map application keys to their identifier space. While this achieves good storage load-balancing and efficient discovery of exact keys, it is in conflict with preserving the semantic proximity as it destroys existing relations among the application-specific keys. Keys which are semantically close at the application level are heavily fragmented in a DHT. We will now briefly present the P-Grid overlay and its range query algorithm. More details can be found in [8].

2.1. The P-Grid Overlay

In P-Grid peers refer to a common underlying binary trie structure to organize their routing tables. Data keys are computed using an order-preserving hash function to generate keys. Without constraining general applicability binary keys are used in P-Grid. Each peer constructs its routing table such that it holds peers with exponentially increasing distance in the key space from its own position. This technique basically builds a small-world graph [12], which enables search in $O(\log N)$ steps. Each peer $p \in P$ is associated with a leaf of the binary trie, i.e., a key space partition, which corresponds to a binary string $\pi(p) \in \Pi$ called the peer's *path*. For search, the peer stores for each prefix $\pi(p,l)$ of $\pi(p)$ of length l a set of references $\rho(p,l)$ to peers q with property $\pi(p,l) = \pi(q,l)$, where $\overline{\pi}$ is the binary string π with the last bit inverted. This means that at each level of the trie the peer has references to some other peers that do not pertain to the peer's sub-trie at that level which enables the implementation of prefix routing.

Each peer stores a set of data items $\delta(p)$. For $d \in \delta(p)$ $key(d)$ has $\pi(p)$ as prefix but it is not excluded that temporarily also other data items are stored at a peer, that is, the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. Moreover, for fault-tolerance, query load-balancing, and hot-spot handling, multiple peers are associated with the same key-space partition (structural replication), and peers additionally also maintain multiple references $\sigma(p)$ to peers with the same path (data replication).

Figure 1 shows a simple example of a P-Grid tree consisting of 6 peers responsible for 4 partitions, e.g., peer F's path is 00 leading to two entries in its routing table: peer E with path 11 at the first level and peer B with path 01 at the second level. Further, peer F is responsible for all data with key prefix 00. A search initiated at peer F for key 100 would first be forwarded to peer E because it is the only entry in F's routing table at level 1*. As peer E is responsible for 11 and not for the key 100, peer E further forwards the query to peer D, which can finally answer the query.

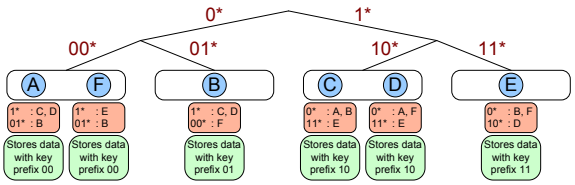


Figure 1. P-Grid overlay network

2.2. Range Queries

Range queries in P-Grid are first forwarded to an arbitrary peer responsible for any of the key space partitions within the range, and then the query is forwarded to the other partitions in the interval using this peer's routing table. The process is recursive, and since the query is split in multiple queries which appear to trickle down to all the key-space partitions in the range, it is called the *shower algorithm*. The intuition of the algorithm is shown graphically in Figure 2.

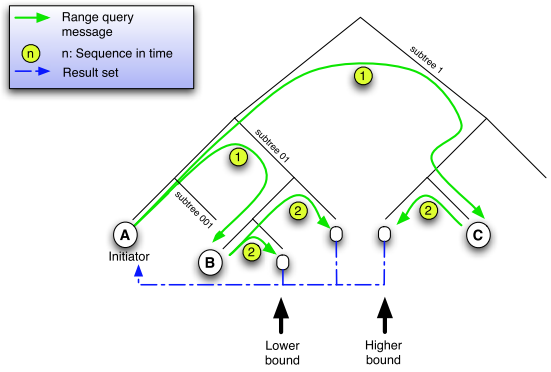


Figure 2. Range query illustration

In the course of forwarding, it is possible that the query is forwarded to a peer responsible for keys outside the range. However, it is guaranteed that this peer will forward the range query back to a key-space partition within the range.

Moreover, the P-Grid routing ensures that no key space partition will get duplicates of the range queries. Algorithm 1 gives the pseudo code for the *shower* algorithm.

Algorithm 1 shower(R, $l_{current}$, p)

```
1: if π(p) ⊆ R then
2:    return(d ∈ δ(p)|key(d) ∈ R);
3: end if
4: determine l_l such that π(min(R), l_l) = π(p, l_l);
5: determine l_h such that π(max(R), l_h) = π(p, l_h);
6: l_min = max(l_current, min(l_l, l_h));
7: l_max = max(l_l, l_h);
8: if l_current < l_max then
9:    for l = l_min to l_max do
10:      r = randomly selected element from ρ(p, l);
11:      shower(R, l+1, r);
12:   end for
13: end if
```

[8] also proposes a more intuitive algorithm which could further be applied to any other structured overlay. The basic idea is to first find the lower (or upper) bound of a range query and then sequentially forward the query along neighbors till the upper (or lower) bound is reached. The simplicity of using only one message has the disadvantages that (i) losing the query message results in the immediate termination of the range query as no further peer is reached and no further data will be returned; (ii) forwarding one message sequentially along neighbors can result in a very long query response time depending on the size of the queried range and the number of peers involved. We will therefore focus on the shower algorithm as its performance is considerable better in terms of response time whereas requiring only a slightly larger number of messages.

2.3. Query Completeness

Though it is guaranteed by the shower algorithm that all peers receive exactly one range query message, it is currently not possible for the initiating peer to estimate the number of peers concerned by a range query, i.e., estimating the number of response messages it has to expect. For keyword based queries, a peer receives only one query response by one peer in a structured overlay network as only one peer (or any of its replicas) is responsible for the given keyword. A peer is therefore able to determine when a query finished and when it received all matching items to either inform a user, start post-processing or initiate subsequent queries. This is currently not possible for range queries in structured overlay networks as the number of response messages depends on the number of peers in the target range, which is usually not know for a peer. We will present in the following section our approach to estimate this number based on the local information available in a peer's routing table and corrected by intermediate peers forwarding range queries or peers responding to range queries. We thereby assume a load-balanced system where each peer holds approximately the same amount of data. Load-balancing in DHTs has been studied intensively as it is one of the main principles of DHTs and [2] presents its realization for P-Grid and evaluation on a real-world test-bed. Hence estimating the number of responding peers is equivalent to estimating the number of query hits expected to be retrieved by a range query.

3. Completeness Estimation

Estimating the completeness of queries should intuitively be bound to the data level: the user is interested in what fraction of all expected result hits he already received. This also holds for subsequent processing steps following the execution of range queries. As briefly mentioned in the last section, predicting completeness on data level is almost impossible without enormous costs. Fortunately, in a load-balanced overlay system this completeness can be mapped to completeness on reply level, because each reply should deliver approximately the same number of results. This is especially true for range queries, because no filtering steps are applied – if a peer is responsible for a part of the range, it will return all of its local data items. Moreover, we will show that we are able to guarantee to identify the last query reply when receiving it. Thus, a completeness of 100% on reply level corresponds to a guaranteed completeness of 100% on data level. So, for subsequent operations that rely on complete range query replies estimation on reply level is absolutely satisfying. In order to show its applicability for other situations, in Section 6 we show that completeness on data level and reply level almost match. Note that, due to the characteristics of sophisticated overlays, the majority of queries will be answered completely.

In the following, we will introduce our basic approach and the principles it is based on. An empirical evaluation as well as thoughts on determining a quality of the estimation in addition are presented in the subsequent sections.

3.1. The Proposed Method

We focus on the mentioned shower algorithm implemented in P-Grid. In Section 4 we discuss the possibilities for other systems to provide completeness estimation for range queries and the applicability of our approach to them.

A peer initiating a range query starts this query by providing the interval bounds of the desired range. Afterwards, each intermediate peer responsible for routing the query, forwards it to one or more sub-trees, depending on its own path, the paths of peers from its routing table, and the paths of the queried range. Thus, the crucial point is to estimate the number of peers responsible for a certain key range. But, due to load-balancing aspects, this is quite difficult. The idea is to use all available path information in order to build an estimated P-Grid trie. Based on this tree, we can determine a minimal number of replies expected.

In the following we will explain, how we can determine the minimal number of replies from an estimated P-Grid trie. Let

$$b_1 b_2 b_3 \dots b_x$$

denote the x bits that form the binary key of such a peer. From this path, we can deduce the existence of at least x other peers: Let $\overline{b}_i$ denote the inverted bit b_i. For each path

1: $\overline{b}_1$

2: $b_1 \overline{b}_2$

3: $b_1 b_2 \overline{b}_3$

...

x: $b_1 b_2 \dots b_{x-1} \overline{b}_x$

there must exist *at least one* responsible peer. Knowing about several paths from peers in a range, the initiator can deduce a minimal number of peers in that range. In order to achieve this, the initiator builds a tree from those paths and reflects to the minimal number of peers.

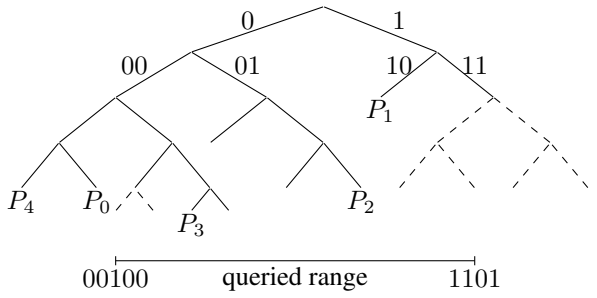


Figure 3. Estimating the P-Grid trie

Figure 3 illustrates this. The figure shows an example P-Grid tree. Assume a query for the range $00100 - 1101$ was initiated. Further, the initiator P_0 knows about four peers, where the paths from P_1, P_2 and P_3 are in the range. As every peer has at least one reference to another peer for each of the positions of its path, P_0 must at least know about four peers, each located in a different sub-tree. The part of the tree the initiator can deduce from its local routing information is shown in solid lines. The dashed lines indicate that part of the tree not known to the initiator, which results in a small error in this first estimate. The minimal number of peers in the range estimated in this situation is 8, the correct value is 10.

3.2. Estimation Refinement

The first estimation performed by the query initiating peer is solely based on the routing information available at that peer. The local routing table stores at least one reference per level. In other words, we know about at least one peer of each sub-tree a range query is sent to. For fault-tolerance and load-balancing reasons structured overlays usually keep multiple references at each level to remain operational during peer churn or to select the least loaded peer for query load-balancing. Therefore, the information a query initiating peer has about the structure and peers in a sub-tree increases with the number of references per level.

But, the information gathered like this is still not complete and the estimation might still be too small as some peers remain "invisible" from the local point of view. Therefore, initiating peers piggy-back with each query sent to a sub-tree the estimate of peers considered in a sub-tree. For example in Figure 3, the range query sent from peer P_0 to peer P_3 also the contains the estimate that three peers build the sub-tree 001*. As P_0 only knows that P_3 has path 00110, it knows that there must be a peer 00111 and at least one peer for 0010, though P_0 does not know that the sub-tree 0010* actually consists of two peers. P_3 is aware of this fact, because P_3's routing table must contain at least one of the peers from sub-tree 0010*, and can return the correct number of peers in sub-tree 001* with its query reply to peer P_0. P_0 can then correct the estimate of query replies expected for the initiated range query. Peers receiving a range query with correct information do not have to "correct" the initial estimate.

The required message overhead for our completeness estimation is therefore minimal as no additional messages have to be sent and only small information are piggy-backed with sent query and query reply messages. In case a range query hits a peer outside the target range with an incorrect estimate, the receiving peer can either react by replying with a short acknowledgment message correcting the initial estimate, or it forwards the incorrect estimate to target peers in the range and the correction will be returned in the query reply messages. In the first case, the query initiator can sooner correct the estimated completeness at the cost of a small extra message, whereas in the second case the correction is done at a later time with the reception of query results without additional messages.

Applying the method as described above, we will never over-estimate the number of expected replies. Moreover, when a query is finished, we will always recognize this for sure. This is possible because the paths of the replying peers are analyzed. Thus, receiving these replies, we always know for sure the actual size of the corresponding sub-tree.

3.3. Further Improvements

There was much research spent on designing overlay systems as much stable and reliable as possible. Thus, we can even cache estimated trees once they are built. These cached trees can later be used for subsequent queries. The trees should then be adapted to changes in the overlay structure registered – which may, of course, occur, but are expected to be rather rare. In this way, we achieve a quite accurate and satisfyingly exact completeness estimation, which is automatically maintained with each query initiated.

The task of achieving complete query results is due to the used overlay system, in this case the P-Grid overlay. Nev-

ertheless, incomplete results may occur in rather unstable and unreliable large-scaled systems. This also effects the completeness estimation, as, for instance, we will experience a difference in the *static* completeness concerning all data that should be available, and the *dynamic* completeness based on the results actually received. This should be involved into completeness considerations. A nice aspect of the method proposed here is that it allows for estimating the size of results missing in this case.

4. Usability in other Overlay Systems

Our approach is based on a parallel resolution of range queries in a binary trie similar to a prefix hash tree, whereby in the case of P-Grid the depth of each sub-tree can be estimated by the known nodes of this sub-tree stored in the local routing table. To the best of our knowledge no other system can already provide completeness estimation for range queries. In this section, we briefly discuss the possibilities for other systems to estimate the number of query replies and the usability of our approach for them.

The approach for range queries in SkipGraphs [4, 3] is the most similar one to the one of P-Grid as peers also maintain routing information at multiple levels. Our proposed method can also be used by SkipGraphs to estimate the number of peers in other sub-trees. The only problem is the number of peers remaining in the bucket layer below the lowest interconnected skip-list level. But, as load-balancing is in place, this number should be similar to the number of buckets the current node is in.

Approaches like [14] and [13] are based on a prefix hash tree where peers remain at each level of the tree, unlike in P-Grid where peers only remain at the leaf level. The routing in this tree starts at the root level and trickles down the tree from nodes to their children until all nodes in the target range are reached. As we assume that nodes do not know the exact number of their children, it is not possible for them to estimate how many nodes will return results for a range query. If this number can be estimated, the technique presented in this paper can also be adapted for completeness estimation in systems based on prefix hash trees.

Finally, approaches forwarding a range query sequentially along neighbors cannot estimate the final number of nodes involved in a range query. This holds for CAN-based systems presented in [9, 15] and the Chord-ring based system Mercury [5].

5. Applying Completeness Estimation

In the last sections, we motivated the need for an exact completeness estimation and proposed a technique for achieving this on the physical layer. Now we will discuss how the introduced method is applied internally and how completeness information are presented to the user, used for subsequent processing steps, respectively.

By applying the method as introduced, we can always determine a minimal value for the number of replies expected. With each reply received, we just compute the ratio

$$\frac{\text{received replies}}{\text{expected replies}}.$$

By this, we are always able to provide a satisfyingly exact estimation and identify the final reply for sure. With the first replies this completeness value may be over-estimated, because we expect less replies than finally received. But, as Section 6 will show, the number of corrections needed for determining an exact estimate is rather low. Moreover, the final completion of a query can always be determined by 100%. Preferably we should also provide a corresponding probability or intervals of (un-)certainty for all intermediate replies.

One idea is to provide an expected average number rather than the minimal number of replies. This can be achieved by using information about the average depth of a P-Grid trie, which could be taken from empirical studies. The minimal number of replies could be used as a lower bound in this case, and an estimated maximal number (again, based on empirical studies) as an upper bound. Like this, we can easily provide a completeness estimate plus an interval of certainty. Of course, the value estimated for the maximal number of replies will rise suddenly when a query reply from an unexpectedly deep sub-tree is received. It is a fact that the quality of estimation is the worse, the more unknown sub-trees we have *and* the higher they are located in the tree, i.e., the shorter the known paths are. This can be even improved if we base the completeness estimations on cached estimation trees. These trees, built with queries already finished, usually reflect a relatively good view of the routing trie. Even if we have no glue about the actual size of the tree, we could still use, if available, an estimated number of peers joining the system in order to determine upper bounds for the number of expected replies and thus, provide goodness intervals.

Another conceivable approach is to compute a kind of error approximation. For instance, we could use the estimated maximal tree depth in order to compute an additional quality estimation. One suggestion for such a factor of uncertainty ϵ_l is:

$$\epsilon_l = 1 - \frac{|B_l|}{|B_l| + \sum_{b \in \overline{B}_l} 2^{b_{max}-|b|}}.$$

B_l is the set of all paths known to the initiator overlapping the queried range, $\overline{B}_l$ represents the minimal set of paths b predicted to be still missing in the range – in other words, the number of sub-trees we know overlapping the range but we still miss any information from. b_{max} stands for the assumed maximal path length/tree depth ($\forall b \in B_l \cup \overline{B}_l : b_{max} \geq |b|$). Assuming $b_{max} = 5$ in the small example from Section 3.1 we would predict a factor of uncertainty $\epsilon = 1 - \frac{3}{3+(2+1+4+2+8)} = 0.85$. On the first look, this

seems to be an unjustified high value – but, at this point, we only know of three peers among at least eight in the range, which is only 37.5%.

As one example for applications that will benefit from the proposed method for completeness estimation we refer to UniStore [11]. UniStore is a light-weight system for universally storing and managing structured data in a distributed manner based on a structured overlay. Range queries are only a part of the variety of query processing techniques supported by UniStore. There is a need to provide exact completeness estimation for all of these types. An example for processing techniques relying on range queries and an exact completeness estimation are skyline queries [6, 10]. One step during the processing of these queries is to initiate range queries for each of the queried attributes – and continuing with subsequent processing steps after all (or a satisfyingly large fraction) range query replies are received. Other supported operators that rely on range queries are aggregates. For instance, the average of an attribute is determined by initializing a range query for the attribute. This is possible because the values are inserted into the storage system by applying a hashing function on the attribute name concatenated with its value. Note that P-Grid uses a prefix-preserving hashing approach, so all items for one attribute are stored in a continuous range.

6. Evaluation

The focus of the following evaluation is to show the applicability, exactness and quality of the proposed completeness estimation. These aspects are not directly depending on the size of the network, but rather on the size of the constructed overlay trie. This, in turn, also but not exclusively depends on the network size. We created a local and reliable but real environment consisting of 61 nodes. These nodes were physically distributed over 20 machines, each running up to 4 instances listening on different ports. As the environment was stable, we were able to use a low replication factor, lowering the number of replicas responsible for one path in the P-Grid trie. This resulted in a wider and deeper tree. Thus, the results are also significant for larger scaled networks, where usually a higher replication factor is used. We used two environments, the first with a replication factor of 2, the second with a factor of 1. In unreliable systems, this factor will be set to 5 or higher compensating frequent joins and leaves of peers. Our evaluation focuses on the completeness estimation of range queries and we assume that P-Grid guarantees the availability of at least one peer per partition even in very dynamic or unreliable setups like PlanetLab.

We inserted 48 data items from each of the peers, resulting in a total of 2928 data items. The used string data taken from IMDB (`http://www.imdb.com/`) represents information about movie titles and shows a skewed heavy-tail key distribution (power-law like). The average number of leafs, maximal path length and the average path length were 32, 8 and 5 for a replication factor of 1. For a factor of 2, the values were 19, 6 and 4.5, respectively. The resulting P-Grid trie was not balanced. Almost 40% of the leafs were located under key prefix 0 and the tree was deeper and wider under key prefix 1.

In order to evaluate the influence of the number of references for one level of the local routing table we built three environments, using a maximal number of references of 1, 3 and 5. A query mix of three different range queries, involving different parts of the trie and therefore resulting in a different number of replies, was run. Query q1 asks for all index entries with prefix 10 (range 100...0-101...1), q2 for all with prefix 11 and q3 uses an empty prefix, thus querying the whole trie. Each query was initiated 10 times, each time on a randomly chosen node. In the following, we present and discuss the results of the described experiments.

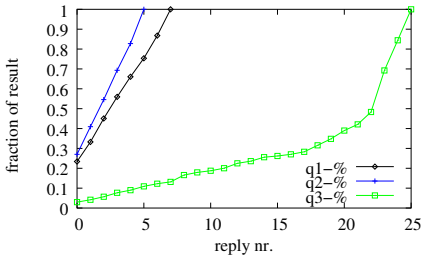
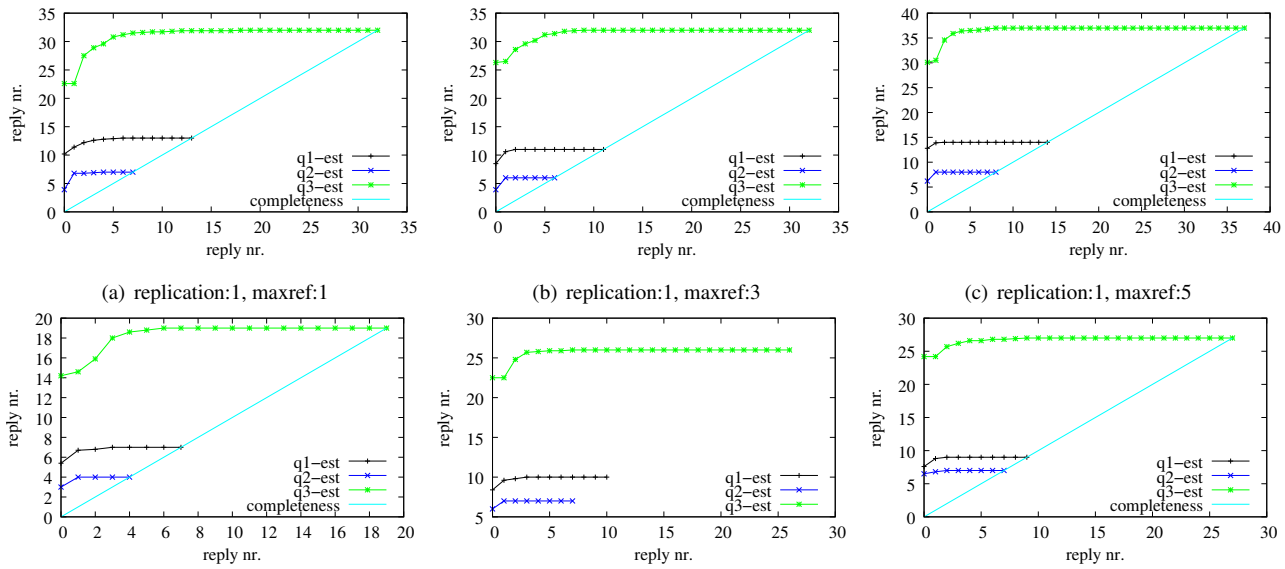


Figure 4. Completeness on data level

The first figure shows the correspondence between completeness on data level and on peer level. Figure 4 shows the percentage of the final result received with respect to the number of replies received. We exemplary chose one of the described network environments (replication factor 2, maximal references 5) – in the other settings results look similar. The plot shows that especially for the two queries resulting in less answers the development of the result size is almost linear. For the query involving the whole P-Grid trie the last query replies contain larger fractions of the result than earlier replies. Even if P-Grid implements a sophisticated load-balancing, there might exist keys a particular high number of data items is mapped to. P-Grid's load-balancing technique splits high frequented key space partitions more fine-granular than others, but does not "split" single keys. Thus, some peers are still responsible for a higher number of items than others. Due to the locally used storage system, the answer time correlates to the amount of data to be processed locally. Therefore, replies from these peers arrive with later replies, resulting in a higher increase of the result size with the final answers. A perfect mapping would be indicated by a straight line. The figure shows that the mapping from completeness on data level to the completeness on reply level is satisfyingly realistic in load-balanced overlay systems.

Figure 5 shows the number of replies we estimated using the proposed technique with each reply received. Addition-



(a) replication:1, maxref:1 (b) replication:1, maxref:3 (c) replication:1, maxref:5

(d) replication:2, maxref:1 (e) replication:2, maxref:3 (f) replication:2, maxref:5

Figure 5. Estimated number of replies with corrections

ally, the straight line represents the actual completeness on reply level. The figure clearly shows that our method always estimates the number of replies correctly at the end. Moreover, it gets evident that only a small number of first replies is needed in order to determine a correct value in the end. As expected, the higher the number of references for each level of the routing table, the more exact the initial estimation and the less corrections are needed. The figure also shows that in this case the size of the temporary errors is smaller than with lower references per level. The differences in the number of replies for identical queries are due to the need for starting networks with different parameters from scratch every time. By this, and the application of a random-walk strategy in order to build the P-Grid trie, this results in, only slightly, different overlay trees.

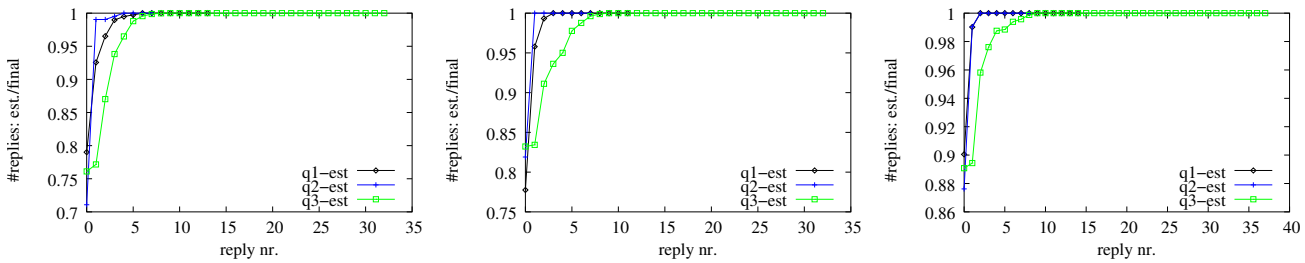
The smaller the part of the trie involved into range query processing the less information is needed in order to achieve exact estimations. For the first two queries, even the settings using a replication factor of 1 and/or a maximal number of references per level of 3 and 1 are quite satisfying. As subtrees are queried more probably than the whole tree, this shows that the proposed method provides quick and exact estimations even with low information. This also indicates the effectiveness for larger scaled and unreliable systems where, in turn, more information shall be contained in the local routing tables.

Summarizing, we can state that for each of the considered cases we only need a fraction of replies in order to achieve an exact completeness estimation. As the method goes along with very low additional effort, this proves its powerfulness for trie structured overlays in general.

The last figures show the relative completeness ($\frac{\text{estimated \#replies}}{\text{final \#replies}}$) estimated with each reply. Thus, it illustrates the ratio of error correction. Moreover, this time the queries were run on two different networks for each setting, each of them run for a different time before starting queries. Results from the hence four runs were averaged. Thus, effects of slightly different overlay tries are eliminated. Figure 6 shows that the ratio of correction is always almost equal for each of the used environments. Following, independent from the query actually initiated, completeness estimation is comparably good and corrections provide equally good improvements with respect to the size of the final result. The figures also show that the initial estimate is good for all tests, but it is better if more references are stored at each routing table level. As expected, an error correction can be recognized only for the first query replies and converges to 0 for later replies. Another important observation is that the estimation for the queries with less replies are very exact with little information and that the corresponding plots approach each other with rising numbers of references.

All in all the proposed method for completeness estimation is absolutely satisfying. The initial estimation, based on no further knowledge than the local one, is quite good for any type of query and environment. Even if this first estimate is erroneous, only a small amount of replies is needed in order to determine an exact estimate.

In the presented experiments, we omitted any dynamics, which is of course a natural ingredient of P2P systems. But, providing complete results under churn is a task the overlay is responsible for, not the system layered on top of it, like UniStore. If the overlay guarantees complete results, which modern overlay systems achieve by applying adequate replication algorithms, the proposed approach will work correct,



(a) maxref:1 (b) maxref:3 (c) maxref:5

Figure 6. Estimated relative number of replies with corrections

as shown above. Moreover, in recent time the super-peer approach, relying on a fraction of all peers which are rather reliable and robust, gains boosted attention. Referring to this, the static network used in our tests corresponds to such robust super-peers. The only point where minor errors may occur is if a peer crashes right after receiving and acknowledging a query, but before replying to it, which is rather unlikely. Nevertheless, we are preparing more complex experiments including dynamics. By this we will evaluate the robustness of the used overlay in combination with completeness estimation for all operators supported by UniStore.

7. Conclusion

In this work, we faced the problem of completeness estimation in structured overlay systems, where knowledge is usually bounded locally. We motivated the need of an exact estimation for both, user and internal requirements. Beside the limited knowledge, the problem of completeness estimation is quite challenging due to the natural aim of the investigated systems to parallelize a query as much as possible. This gets especially obvious for efficient approaches of range query processing. The proposed method solves this task for several system architectures, while being low-effort but though exact. Thus, it represents an efficient and powerful technique for modern distributed data- and information systems.

Main aspects of our ongoing work are the integration of the approach into a sophisticated completeness estimation supported by complex systems like UniStore and the investigation of further algorithmic improvements as well as an extended analytical and empirical evaluation. Currently we are developing measures and formulas suited to determine quality and goodness of the estimation.

Acknowledgments

We would like to thank Thomas Kreyling for his valuable contribution to the development of a basic approach and his implementation work.

References

[1] K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *CoopIS'01*, pages 179–194, 2001.

[2] K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing data-oriented overlay networks. In *VLDB'05*, pages 685–696, 2005.

[3] J. Aspnes, J. Kirsch, and A. Krishnamurthy. Load balancing and locality in range-queriable data structures. In *ACM PODC*, pages 115–124, 2004.

[4] J. Aspnes and G. Shah. Skip graphs. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, January 2003.

[5] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. *SIGCOMM Computer Communication Review*, 34(4):353–366, 2004.

[6] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE'01*, pages 421–432, 2001.

[7] B. Daniel, P. Hurley, R. Pletka, and M. Waldvogel. Bringing efficient advanced queries to distributed hash tables. In *Local Computer Networks (LCN'04)*, pages 6–14, 2004.

[8] A. Datta, M. Hauswirth, R. Schmidt, R. John, and K. Aberer. Range queries in trie-structured overlays. In *P2P'05*, pages 57–66, 2005.

[9] A. Gupta, D. Agrawal, and A. E. Abbadi. Approximate range selection queries in peer-to-peer systems. In *CIDR'03*, 2003.

[10] M. Karnstedt, J. Müller, and K. Sattler. Cost-Aware Skyline Queries in Structured Overlays. In *ICDE Workshop on Ranking in Databases (DBRank'07)*, pages 285–288, 2007.

[11] M. Karnstedt, K. Sattler, M. Richtarsky, J. Müller, M. Hauswirth, R. Schmidt, and R. John. UniStore: Querying a DHT-based Universal Storage. In *ICDE'07 Demonstrations Program*, pages 1503–1504, 2007.

[12] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. Technical Report 99-1776, Cornell Computer Science, October 1999.

[13] C. Y. Liau, W. S. Ng, Y. Shu, K.-L. Tan, and S. Bressan. Efficient range queries and fast lookup services for scalable p2p networks. In *DBISP2P'04*, pages 93–106, 2004.

[14] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief announcement: Prefix hash tree. In *ACM PODC*, page 368, 2004.

[15] O. D. Sahin, A. Gupta, D. Agrawal, and A. E. Abbadi. A peer-to-peer framework for caching range queries. In *ICDE'04*, page 165, 2004.